VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

Dominik Endrych

Bakalářská práce

Vedoucí práce: Ing. David Seidl, Ph.D.

Ostrava, 2021

Abstrakt

Tato práce pojednává o mé odborné praxi ve firmě Tasty Air s.r.o. V průběhu praxe jsem se zabýval programováním a tvorbou 3D hry v herním enginu Unity na pozici juniorského vývojáře. V práci se nejprve stručně zabývám využitím herních enginů v dnešní době. Dále pak popisuji postup vývoje a jednotlivé komponenty enginu Unity, se kterými jsem se na praxi setkal.

Klíčová slova

Unity; herní engine; odborná praxe; firma; Tasty Air

Abstract

This thesis describes my professional practice in Tasty Air s.r.o. company. During my practice I was involved in programming and development of 3D game in game engine Unity as a junior developer. In this thesis I first briefly describe current usage of game engines. Then I talk about course of development and each component in engine Unity, that I came across during my professional practice.

Keywords

Unity; game engine; professional practice; company; Tasty Air

Poděkování

Rád bych na tomto místě poděkoval především panu Ing. Davidu Seidlovi, Ph.D. za velmi vstřícný přístup a spoustu užitečných rad při vypracování této práce. Dále bych rád poděkoval celému týmu firmy Tasty Air s.r.o. za to, že mi umožnili praxi vykonat a vždy si na mě udělali čas.

Obsah

Seznam použitých zkratek a symbolů

2D	–	Dvojrozměrný prostor
3D	–	Trojrozměrný prostor
AI	–	Umělá inteligence
API	–	Application Programming Interface
AR	–	Rozšířená realita
FPS	–	Střílečka z pohledu první osoby
HDRP	–	High Definition Render Pipeline
UI	–	Uživatelské rozhraní
URP	–	Universal Render Pipeline
VR	–	Virtuální realita

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem této práce je popsat mou praxi ve společnosti Tasty Air s.r.o. a seznámit s dosaženými výsledky. Pro absolvování odborné praxe jsem se rozhodl z důvodu, abych si vyzkoušel práci v profesionálním prostředí a ověřil si tak své dosažené znalosti. Chtěl jsem získat rady od lidí přímo z mého oboru, vyzkoušet si nové zajímavé technologie a připravit se na to, co mě v budoucím zaměstnání čeká.

Po dobu svého studia jsem měl za cíl více proniknout do světa vývoje her a hledal jsem tak pro svou praxi firmu, která by se touto oblastí zabývala. V Tasty Air s.r.o. mi nabídli možnost vyzkoušet si práci v herním enginu Unity, což je momentálně jeden z nejrozšířenějších herních enginů vůbec. Jednalo se o skvělou příležitost, jak zjistit, jestli by mě tato práce skutečně naplňovala.

Tato práce pojednává o mém nasazení ve firmě Tasty Air s.r.o., kde jsem dostal za úkol postavit prototyp 3D hry úplně od základu. Na tomto projektu jsem měl příležitost vyzkoušet si velkou řadu možností herního enginu Unity a naučit se dobré praktiky tohoto enginu. Dále jsem se dozvěděl, jak se postupuje při vývoji her (ale i VR a AR aplikací) v profesionálním prostředí.

Ve 3. kapitole se objeví srovnání dnes asi nejpopulárnějších herních enginů. Další kapitoly se budou zabývat již samotným vývojem hry úplně od základu. Všechny důležité nástroje a pojmy budou vysvětleny tak, jak byly při vývoji použity. Na závěr této práce jsou zhodnoceny dosažené cíle a vypíchnuty některé nedostatky s možným vylepšením.

Kapitola 2

Seznámení s firmou Tasty Air s.r.o.

2.1 O společnosti

Firma Tasty Air s.r.o. vznikla na začátku roku 2019 a jedná se tak o poměrně mladou společnost se sídlem v Ostravě Vítkovicích. Společnost však založili lidé, kteří se s profesionálními aplikacemi setkávají již přes mnoho let a jedná se tak o prostředí na velmi vysoké úrovni. Již od svého vzniku se firma naplno věnuje aplikacím pro virtuální a rozšířenou realitu. Klientům nabízí nové možnosti školení svého personálu, virtuální prezentaci netradičních prostředí nebo třeba uchování kulturního dědictví v prostředí virtuální reality. Jelikož se jedná o menší firmu, je všechen vývoj vysoce orientován právě na klienty, kterým nabízí řešení přímo na míru podle jejich potřeb. Díky své bohaté historii ve vývoji her navíc dokáží udělat svá řešení uživatelsky přívětivé pro všechny generace. [5]

V České republice je Tasty Air s.r.o. jednou z mála společností, která si zvolila virtuální a rozšířenou realitu jako hlavní pole své působnosti. Díky zaměření na tuto stále méně rozšířenou oblast již spolupracovali s řadou partnerů po celém světě a to i přes to, že jde o firmu spíše menšího charakteru.

2.2 Zařazení ve firmě

Na praxi jsem byl přijat po pohovoru a ukázce mé dosavadní školní práce. Byla mi přidělena juniorská pozice Unity vývojáře. Jako zadání jsem dostal projekt, jehož hlavním účelem bylo, abych se podrobně seznámil s herním enginem Unity a naučil se jej používat i v profesionálním prostředí. V období své praxe jsem měl vytvořit funkční prototyp 3D hry se survival prvky.

Veškeré instrukce o tom, co vše má hra obsahovat a jak by měla vypadat byly podrobně sepsány v design dokumentu. Mým úkolem bylo postupně vypracovávat jednotlivé prvky hry a držet se časového rozvrhu, na kterém jsme se s vedoucím ve firmě domluvili. Celou mou práci jsem pak vždy konzultoval a ostatními členy týmu. Zkušenější Unity vývojáři vždy zhodnotili mé řešení a udělili mi spoustu cenných rad.

Kapitola 3

Současný stav herních enginů

Hry se v dnešní době staly již nedílnou součástí našich životů. To vedlo mimo jiné i ke vzniku různých nástrojů a novinek, které herní vývojáři využívají na denní bázi. Dříve bylo nezbytné, aby si programátoři vytvořili svůj vlastní herní engine, což může být velmi časově i finančně nákladné. Dnes však mají možnost využít některý z již ověřených enginů třetích stran. Tyto nově vzniklé herní enginy pak více otevřely trh hlavně nezávislým a amatérským vývojářům.

Jelikož se na trhu nyní objevuje stále více herních konzolí a mobilních telefonů, začaly se programátoři ubírat směrem vývoje her na více platforem najednou. Autoři již zmíněných enginů tak museli změnit své nástroje tím způsobem, aby tuto poptávku uspokojili.

Herní enginy navíc již nejsou pouhým nástrojem pro vývoj her, ale našly využití i v mnoha jiných oblastech – virtuální školení, virtuální prezentace nebo simulační software. Tento trend ještě více posílil nástup virtuální reality.

3.1 Herní engine

Herní engine hraje velmi důležitou roli v naprosto každé hře, protože pomáhá vývojářům oživit postavy, ovládat scény, vykreslovat grafiku nebo spouštět animace. Herní engine se dá považovat za prostředí, které vývojářům nabízí sadu nástrojů a komponent pro vývoj. Herní enginy otevřely svět vývoje her zejména menším vývojářům, protože jim nabízí vrstvu již funkčních komponent a programátoři se tak mohou zaměřit na tvorbu logiky samotné hry.

Požadavkem na herní engine je v dnešní době stále více jeho dlouhá životnost. S rychlým rozvojem technologií je však tento cíl mnohdy těžko dosažitelný. Dobrým příkladem může být multiplatformní vývoj, na který se klade stále větší důraz. Při zavedení nové platformy pak musejí vývojáři enginu myslet na to, aby uživatel enginu mohl plynule přejít na vývoj pro novou platformu a zároveň tím nezpomalil vývoj na platformu jinou.

Herní enginy dnes obsahují pět velmi důležitých komponent [1]:

- **Hlavní herní program** – obsahuje samotnou herní logiku

- **Renderovací engine** – generuje veškerou grafiku a animace
- **Audio engine** – algoritmy, které zajišťují zvukovou funkcionalitu
- **Physics engine** – zajišťuje ve hře funkčnost "fyzikálních“ zákonů
- **AI modul** – nástroj, který může vývojář využít pro tvorbu umělé inteligence

3.2 Srovnání nejpopulárnějších herních enginů

3.2.1 Unity

Unity Engine vzniká už od roku 2005 a od té doby se stal velkým pomocníkem nespočtu autorů nezávislých her. Na starost jej má společnost Unity Technologies. Hlavní výhodou tohoto enginu je množství multiplatformních balíků, které umožňují vývoj her na většinu známých herních platforem. Unity Engine je vhodný pro 2D i 3D tvorbu a dále je velmi dobrou volbou pro vývoj VR a AR aplikací. Scripty v tomto enginu jsou tvořeny pomocí jazyka C#, který je velmi flexibilní a poměrně rychlý, což z něj dělá pro vývoj her dobrou volbu.

Při použití Unity Enginu má vývojář možnost využít poměrně rozsáhlého obchodu s Assety, kde najde jak placené, tak neplacené assety. Licencování tohoto enginu je výhodné zejména pro menší vývojáře, protože ti jej mohou používat naprosto zdarma. Vývojáři jsou povinni zaplatit si licenci až ve chvíli, kdy jejich příjmy z vyvíjené hry přesáhnou 100 tisíc dolarů.

Unity Engine má však i několik nedostatků. Jak již bylo zmíněno, profesionální vývojáři a větší studia musejí platit poměrně drahou licenci. Tento engine také může být dosti náročný na počítač a při větším množství využitých nástrojů zabírá na disku skutečně dost místa. Poslední malá nevýhoda pak pramení z častých updatů, které v některých případech mění UI editoru. [2]

3.2.2 Unreal Engine

Unreal Engine je herní engine, který byl poprvé představen v roce 1996 společností Epic Games. Nejprve byl zamýšlen jako engine na vývoj FPS, ale s rostoucí popularitou se jeho pole působení rozšířilo. Největší silou tohoto enginu je to, že zvládá skutečně špičkovou grafiku. Obsahuje velkou řadu nástrojů pro detailní práci s materiály, vizuálními efekty a herními levely. Díky těmto vlastnostem se stal rozšířenou volbou pro velká herní studia. Skvělou volbou je pak i pro vývoj VR aplikací. K tvorbě chování hry využívá velké řady nástrojů pro vizuální programování (blueprinty) a dále pak jazyk C++.

I tento engine se může pyšnit poměrně rozsáhlým obchodem. Unreal Engine je navíc open source, což znamená, že se na jeho rozvoji podílí i komunita externích vývojářů. Hlavním nedostatkem tohoto enginu může být jeho náročnost na hardware. Díky svého zaměření na špičkovou grafiku vyžaduje tento engine silný počítač i u menších projektů. Dále je pak nutno podotknout, že tento engine sice podporuje vývoj 2D her, ale stále je zaměřen spíše na vývoj 3D a VR her. [2]

3.2.3 Godot

Herní engine Godot byl poprvé představen v roce 2014 a od té doby si pomalu získává na popularitě. Tento engine je vhodný zejména pro ty, kdo hledají engine, který je zdarma a open-source. Godot podporuje vývoj 2D i 3D her. Jeho licence je naprosto zdarma a lze ji využít i při profesionálním vývoji. Jako poměrně nový engine má však ještě spoustu nedostatků, které pramení hlavně z jeho krátké doby existence. Na rozdíl od již zaběhnutějších enginů, Godot nemá tak velkou komunitu a nabízí menší obchod. Pro vývoj her navíc používá svůj vlastním skriptovací jazyk GDScript, který se podobá jazyku Python. [2]

Kapitola 4

Postup vypracování zadaného úkolu

4.1 Založení projektu

Ještě před začátkem práce na projektu samotném bylo potřeba určit, jaká výchozí nastavení bude projekt mít. Tato volba vychází hlavně z toho, jak má být finální produkt graficky náročný a jaké grafické operace bude potřeba provádět. Na základě těchto informací se pak zvolí vhodný vykresovací nástroj, tzv. *Render Pipeline*.

Render Pipeline provádí serii operací, které vezmou obsah scény a vykreslí je na obrazovku. Mezi tyto operace patří třeba vykreslování nebo aplikování vizuálních efektů. [3]

Jelikož může být obtížné uprostřed vývoje změnit renderovací pipeline, je dobré si tuto volbu rozmyslet dopředu. Různé volby mohou mít rozdílné vlastnosti shaderů a další nástroje. Unity podporuje i možnost vytvoření a úpravu vlastní Render Pipeline, ale to je časově náročné a proto je většinou lepší vybrat si jednu z možností, které Unity Engine momentálně nabízí.

4.1.1 High Definition Render Pipeline

High Definition Render Pipeline (dále jen HDRP) se zaměřuje pouze na výkonný hardware, tedy PC a konzole PlayStation či Xbox. Je určena zejména pro vývoj her s komplexní a vysoce náročnou grafikou. HDRP podporuje velkou řadu grafických technik, díky kterým může vývojář vytvořit velmi propracované materiály, textury a scény. To však přichází s cenou vysoké časové náročnosti. Pro efektivní využití HDRP je pro každý materiál potřeba vytvořit hned několik druhů texturových map (např. metallic map, smoothness map, detail map, normal map atd.). Pro vývoj hry s méně náročnou grafikou a efekty tak může být využití HDRP zbytečné.

Největší výhodou HDRP je určitě osvětlení, protože nabízí mnohem více světelných možností.

- Globální osvětlení v reálném čase – simuluje odrazy světla
- Volumetrické osvětlení – simuluje průchod světla přes částice ve vzduchu

- RayTracing – momentálně úplně nový způsob, jak simulovat světlo, odrazy a stíny
- Podpovrchový rozptyl – průchod světla přes tenké průsvitné objekty

4.1.2 Universal Render Pipeline

Universal Render Pipeline (dále jen URP) je vytvořena tak, aby měla výsledná hra dobrý výkon na téměř jakékoli platformě. Pokud tedy hra nemá speciální nároky na vizuály, které nabízí pouze HDRP, je velká šance, že tato Render Pipeline bude lepší volbou. Už od základu nabízí většinu toho, co dokáže HDRP. V rámci podpory více platforem však bylo několik funkcí ořezáno nebo úplně odstraněno.

URP oproti HDRP dominuje hlavně ve 2D prostoru, protože podporuje 2D světla a stíny. Pro vývoj 2D her je tedy URP žádoucí.

V této práci se zabývám hrou, která bude mít jednoduchou grafiku a nepotřebuje složité efekty. Z těchto důvodů tedy byla zvolena URP. Podrobnější srovnání obou zmíněných možností lze vidět v tabulce 4.1.

Vlastnost	HDRP	URP
Shaderový graf	Ano	Ano
Graf vizuálních efektů	Ano	Ano
VR	Ano	Ano
RayTracing	Ano	Ne
Volumetrické osvětlení	Ano	Ne
Globální osvětlení v reálném čase	Ano	Ne
Plošné světlo	Ano	Ne
Podpovrchový rozptyl	Ano	Ne
Skládaný shader	Ano	Ne
Ambient Occlusion	Ano	Ne
Automatická expozice	Ano	Ne
Odrazy světla	Ano	Ne
Skládání kamer	Ano	Ano
Fyzické kamery	Ano	Ano
Záře žárovek	Ano	Ne
Odlesk objektivu	Ano	Ne

Tabulka 4.1: Srovnání vlastností HDRP a URP

4.2 Využití skriptů v enginu Unity

Skriptování je důležitou součástí všech aplikací v enginu Unity. Většina aplikací využívá skripty k reakci na input od hráče nebo řetězení herních událostí. Skripty navíc mohou vytvářet grafické efekty, ovládat chování objektů nebo implementovat vlastní umělou inteligenci pro všechny postavy ve hře. [3]

4.3 Pohyb hráče

První velmi důležitou funkčností hry byl pohyb hráče. Hra se má hrát z prvního pohledu, což znamená, že hráč model hlavního hrdiny vůbec neuvidí a neměl jsem tím pádem potřebu zabývat se animacemi, které se pojí s pohybem. Tento úkol jsem si rozdělil na dvě části – pohyb těla (ovládán na klávesnici) a pohyb kamery (ovládán myší). Před samotným programováním jsem vytvořil velmi jednoduchou scénu, kde se hráč mohl pohybovat a položil do prostoru několik překážek, na kterých bylo možné otestovat funkčnost. Hráč neměl procházet zdí, propadat se zemí, šplhat na moc strmý kopec a měla na něj působit fyzika.

4.3.1 Funkce komponentu Rigidbody

Rigidbody je jeden ze základních komponentů Unity Enginu. Tento komponent dokáže simulovat reálný pohyb založený na fyzice. Po přidání tohoto komponentu se objekt začne pohybovat díky Unity Physics enginu. I bez přidání žádného kódu bude tento objekt tahán směrem dolů gravitací a bude reagovat na ostatní objekty, pokud je mu přiřazen také Collider komponent. Rigidbody má také API, které dovoluje na jeho objekt aplikovat síly a kontrolovat fyziku realistickým způsobem. Např. chování auta může být specifikováno pomocí aplikace sil na kola. Díky této informace pak může physics engine zpracovat většinu dalších aspektů pohybu auta tak, aby realisticky zrychlovalo a správně reagovalo na kolize. [4]

Při využití Rigidbody komponentu Unity dokumentace doporučuje využívat na operace spíše funkci *FixedUpdate*. Updaty založené na fyzice se totiž provádějí v přesně daných časových intervalech, které nemusejí být ve shodě s výslednými snímky za vteřinu. FixedUpdate se volá těsně před každým fyzikálním updatem a každá změna je tak hned zpracována. [4]

4.3.2 Pohyb hráče do stran

Při řešení první části úkolu, tedy pohybu z místa na místo, jsem měl dvě možnosti. První variantou bylo založit pohyb na komponentu Rigidbody. Při stisknutí patřičných kláves by pak pomocí tohoto komponentu byly na hráče aplikovány síly, které by jej tlačily určitým směrem a tím se postava pohybovala. Při využití Rigidbody komponentu pak dále odpadá veškeré programování fyziky, protože Unity Engine toto chování díky komponentu Rigidbody simuluje. Jak jsem ale zjistil, díky simulaci

fyziky pak nastává situace, kdy hráč může vyšplhat i po velmi strmém kopci. Tento problém by šel vyřešit pomocí vlastního skriptu, ale to by zabralo poměrně dost času. Přistoupil jsem tedy na druhou možnost. Unity Engine alternativně nabízí komponent *CharacterController*.

CharacterController dovoluje jednoduchý pohyb omezený kolizemi bez nutnosti využití Rigidbody. Na CharacterController nepůsobí síly a pohybuje se jen po zavolání funkce Move. Až poté vykoná pohyb, který bude stále omezen kolizemi. [4]

Tento komponent nabízí zajímavou škálu vlastností, které může vývojář ve svém projektu využít. Pro mě bylo výhodné zejména využití vlastnosti, která omezuje, po jak strmém terénu se může hráč ještě pohybovat. Na druhou stranu však tento komponent vůbec neřeší fyziku. Připnutí komponentu Rigidbody na objekt s komponentem CharacterController by však mohlo působit problémy.

Jak je vidět ve výpisu 4.1, Unity nabízí řešení pomocí inputu na horizontální a vertikální ose. Jak již bylo řečeno, díky komponentu CharacterController jsem mohl ovládat pohyb pomocí funkce Move. Tato funkce požaduje parametr typu Vector3, který bude představovat směr pohybu. Tento Vector3 jsem složil právě díky hodnotám na horizontální a vertikální ose. Výsledný vektor jsem pak vynásobil rychlostí a proměnnou Time.deltaTime, která mi zaručila, že pohyb nebyl závislý na snímkách za sekundu

```
Vector3 move = transform.right * Input.GetAxis("Horizontal") + transform.forward *
    Input.GetAxis("Vertical");

characterController.Move(move * speed * Time.deltaTime);
```

Listing 4.1: Ukázka kódu funkce pro pohyb hráče

Pro vyřešení problému s volným pádem bylo potřeba zajistit, aby se objekt hráče začal pohybovat dolů, pokud nestojí na zemi. Pro detekci, jestli hráč stojí na zemi, jsem na úplný spodek objektu hráče připnul další prázdný objekt. Tento objekt pak kolem sebe pomocí funkce Physics.checkSphere vykreslí malou kouli a vrátí, jestli tato koule kolidovala s objektem, který se shoduje s maskou 'Ground.' Pokud tato kolize nenastala začne se objekt hráče pohybovat dolů a postupně nabírat rychlost. Jakmile opět koliduje se zemí, je tato rychlost vrácena na původní hodnotu.

4.3.3 Pohyb kamery

Pro větší přehlednost kódu jsem pro pohyb kamery vytvořil úplně nový skript, který byl připnut na objekt hráče. Z kamery jsem udělal potomka hráčova objektu, protože se má pohybovat pouze lokálně v závislosti na tom, kde se hráč zrovna nachází. K získání pohybu myši na ose X a Y nabízí Unity Engine opět hodnoty na předdefinovaných osách. Jedná se o osy "Mouse X"a "Mouse Y."

Při pohybu podle osy X jsem potřeboval, aby se kamera neotáčela, ale místo toho se otočilo celé tělo hráče. Vytvořil jsem tedy proměnnou typu Quaternion, která představovala cíl, jak moc

by se mělo hráčovo tělo otočit. Po získání této proměnné jsem pak aplikoval rotaci na komponent Transform hráčova objektu.

Pro pohyb podle osy Y jsem naopak potřeboval, aby se otáčela pouze kamera a hráčovo tělo nebylo nijak ovlivněno. Nejprve jsem tedy získal hodnotu na předdefinované ose "Mouse Y"a následně ji ještě oříznul tak, aby se hlava mohla otočit pouze v určitém intervalu mezi dvěma úhly. Tato rotace však byla aplikována na lokální osy kamery, aby byl výsledný cíl pohybu shodný s tím, kam se zrovna hráč kouká na ose X.

4.4 Interakce s předměty v herním světě

Interakce s předměty ve světě měla probíhat tak, že hráč na předmět namíří, předmět se zvýrazní a následně se po stisknutí interakční klávesy provede samotná interakce. Jelikož je tento postup pro všechny předměty stejný, vytvořil jsem vlastní skript Interactable, který se bude starat o všechny interakce. Z tohoto skriptu pak dědí již konkrétní předměty, které si logiku po stisknutí interakčního tlačítka řeší odlišně.

Každý objekt, se kterým je možné provést interakci, má přidělen tag "Interactable."Mnou vytvořený skript pak ve funkci Update vysílá Raycast ve směru, kam zrovna hráč míří. Pokud tento Raycast narazí na objekt s tagem "Interactable,"zvýrazní jej obtažením žlutou barvou. Pro toto vytažení jsem po konzultaci s jedním z programátorů využil skript stažený přímo z Asset Storu.

4.5 Tvorba logiky inventáře

Inventář byl jednou z velkých částí tohoto projektu. Aby měl uživatel poměrně velkou volnost, musel jsem vyřešit několik požadavků:

- Sbírání předmětů
- Skládání předmětů na hromádky
- Přesouvání předmětů v rámci inventáře
- Vyhození předmětu z inventáře
- Konzumace předmětů
- Výroba a skládání nových předmětů
- Nasazení oblečení a zbraní

4.5.1 Prefab systém v enginu Unity

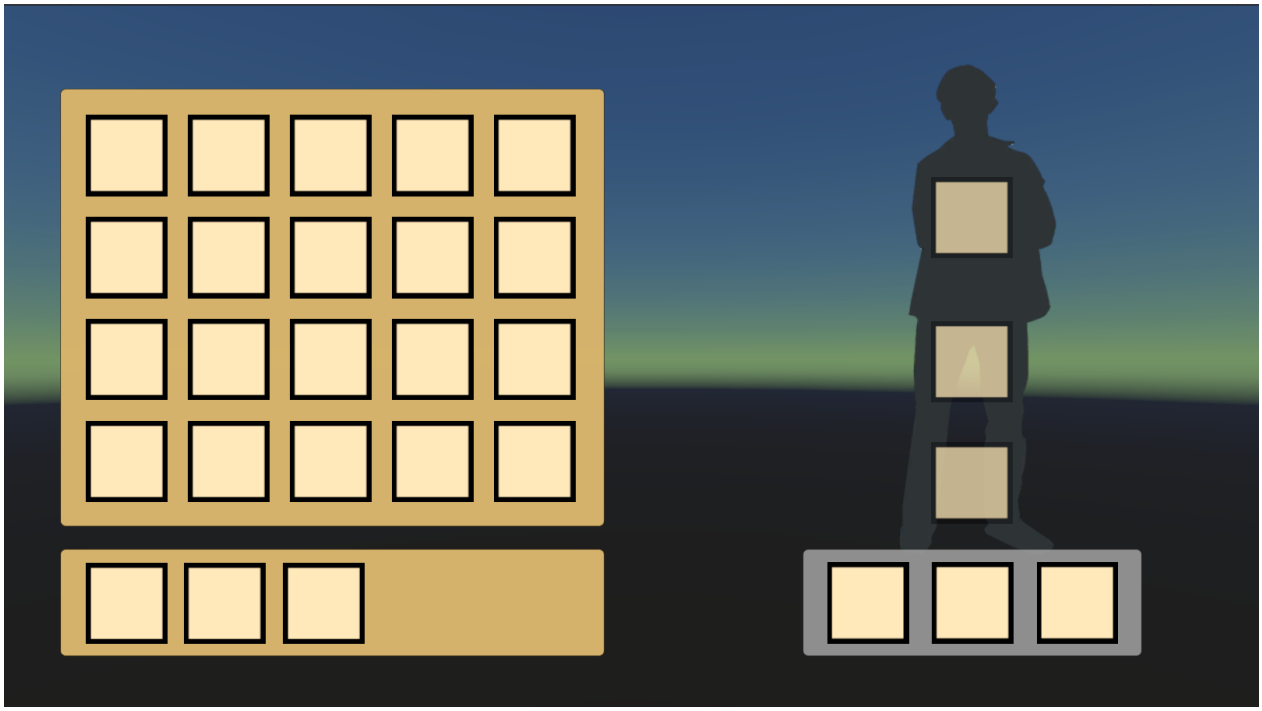
Prefab systém slouží k vytváření vzorů, podle kterých je možné pomocí skriptů vytvářet instance. Výhoda oproti kopii je v tom, že pokud se změní Prefab, změní se i všechny instance. [6]

Vytvořit Prefab je ve většině případů výhodnější než objekty pouze kopírovat, protože Prefab systém dovoluje udržovat všechny instance synchronizované. Dále je však možné modifikovat a přepisovat i jednotlivé instance. Prefab systém je pak vhodné využít, pokud chce programátor vytvářet objekty přímo za běhu hry. [3]

4.5.2 Grafické rozhraní inventáře

Jako první jsem začal tvořit prototyp grafického rozhraní. Inventář se měl skládat celkem ze čtyř částí – jedna představovala samotný inventář, druhá sloty na výrobu předmětů a třetí sloty pro hráčovo oblečení a zbraně. UI se v Unity Enginu tvoří pomocí elementu *Canvas*. Tento element je vždy vykreslen až po vykreslení scény.

Vytvořil jsem nový Canvas a na ten přidal panel, který slouží jako pozadí hlavní inventářové plochy. Na tento panel jsem pak musel umístit všechny inventářové sloty. Přidal jsem tedy do panelu jeden obrázek čtverečku, který slouží jako provizorní grafika pro jeden inventářový slot. Pro budoucí hromadnou úpravu všech slotů jsem z tohoto obrázku udělal Prefab. Na samotný panel jsem pak přidal komponent *Grid Layout Group*, který slouží k uspořádání objektů do mřížky na určité ploše. Výslednou podobu prototypu grafického rozhraní inventáře lze vidět na obrázku 4.1.



Obrázek 4.1: Uživatelské rozhraní inventáře

4.5.3 Využití kontejneru ScriptableObject

ScriptableObject je datový kontejner, který může být použit k uložení velkého množství dat, nezávisle na instanci. Jedním z hlavních použití pro ScriptableObject je zredukování zabrané paměti tak, že se dá vyhnout kopiím hodnot. Na rozdíl od běžných skriptů, ScriptableObject nemůže být připnut na žádný GameObject. Místo toho se ukládají jako samostatné assety v projektu. [3]

Vždy, když se ve scéně instancuje nějaký Prefab, musí být alokována paměť pro všechny jeho držené hodnoty. V případě využití ScriptableObject se však všechny instance pouze odkazují na tento objekt a tím výrazně šetří paměť. ScriptableObject lze upravovat přímo v editoru.

4.5.4 Reprezentace předmětů v kódu

Ještě před ukládáním předmětů do inventáře jsem musel přijít na způsob, jak budou předměty reprezentovány v kódu. K tomu jsem si vytvořil ScriptableObject, který jsem nazval Item. Každému Itemu jsem dal v základu nějaký název, popis, ikonku pro reprezentaci v inventáři a Prefab objektu ze scény, který tento Item představuje. Dále jsem vytvořil další Scriptable Object, tentokrát nazvaný StackItem. Tato třída dědí z původní třídy Item a představuje předměty, které jdou v inventáři skládat na hromádku.

4.5.5 Funkce inventářového slotu

Pro hromadnou úpravu všech inventářových slotů najednou jsem mohl využít již dříve vytvořený Prefab. Na celý slot jsem připnul vlastní InventorySlot skript. Tento skript obsahoval pomocné proměnné pro práci se slotem. Kód asi nejpoužívanější metody AddItem lze vidět ve výpisu 4.2. Dále tento skript implementoval rozhraní IPointerClickHandler, abych mohl zachytit kliknutí na slot. Pokud bylo možné Item ve slotu zkonzumovat (nebo jinak použít), stalo se tak po dvojkliku na slot.

Přesouvání předmětů ze slotu na slot myší jsem pro větší přehlednost rozdělil na dva skripty DragHandler a DropHandler. Tyto skripty implementovaly rozhraní pro detekci přesunu předmětů.

```
public void AddItem(Item newItem)
{
    item = newItem;
    count++;
    icon.sprite = newItem.icon;
    icon.enabled = true;
    isOccupied = true;
    if(!(item is StackItem)){ isFull = true; }
}
```

Listing 4.2: Funkce pro přidání předmětu na inventářový slot

4.5.6 Výroba nových předmětů v inventáři

Pro výrobu předmětů slouží panel pod hlavním inventářovým panelem. Můj úkol zněl tak, že každou možnou výrobní kombinaci měl představovat jeden Scriptable Object. Po nalezení správné kombinace pak hra zobrazila hráči tlačítko a výsledný Item, který si mohl vyrobit.

Vytvořil jsem nový Scriptable Object, který jsem nazval Recipe. Každá instance tohoto objektu pak má kolekci Itemů. Tato kolekce představuje Itemy potřebné na realizaci receptu. Dále má objekt jeden Item, který ve výsledku hráč dostane.

Na samotný výrobní panel jsem připnul skript, který obsahoval List všech možných receptů. Při každé změně na jednom z výrobních slotů projde celou tuto kolekci a hledá, jestli se hráčem zvolená kombinace náhodou neshoduje s nějakou kombinací Itemů ve známých receptech. Část kódu pro nalezení správného receptu je zachycena ve výpisu 4.3.

```
foreach(Recipe recipe in recipes)
        {
            //find recipes that feature same amount of items
            if(itemList.Count == recipe.items.Length)
            {
                if (CheckRecipeItems(recipe))
                {
                    AddItem(recipe.result);
                    if (Inventory.instance.CanPlaceItem(recipe.result))
                    {
                        errorText.gameObject.SetActive(false);
                        craftButton.gameObject.SetActive(true);
                        craftButton.onClick.RemoveAllListeners();
                        craftButton.onClick.AddListener(() => CraftItem(recipe.
                            result));
                    }
```

Listing 4.3: Útržek funkce pro nalezení odpovídajícího receptu

4.5.7 Systém zbraní a oblečení

Pokud má hráč v inventáři nějakou zbraň (na blízku i na dálku), může ji přesunout na jeden ze tří zbraňových slotů. Přesun zbraně na slot jsem již vyřešil v předešlých úkolech. Jelikož však na zbraňový slot mohou být umístěny pouze zbraně, musel jsem skript pro inventářový slot trochu upravit. Přidal jsem proměnnou typu Enum, která určovala, jaký druh Itemu může být na slot položen. V případě hlavní inventářové plochy byly sloty nastaveny na "All"a zbraňové sloty pouze

na "Weapon."Díky této logiky jsem mohl velmi jednoduše udělat i sloty, na které lze položit oblečení. Každé oblečení reprezentuje ClothigItem, což je opět Scriptable Object, který dědí z objektu Item.

Součástí tohoto úkolu bylo vytvořit také to, aby hráč uchopil zbraň přímo do ruky. Jelikož má hráč možnost využít pro zbraně 3 sloty, měl jsem za úkol udělat logiku, díky které by mezi těmito zbraněmi přepínal pomocí číselných kláves 1-3 s tím, že uvidí i model zbraně. Momentálně jsem nemusel řešit animace, ale vedoucí mi doporučil, abych si alespoň jednoduchou animaci později zkusil.

K tomuto úkolu jsem dostal jednoduchý model revolveru (zbraň na dálku) a na testování jsem si sám vytvořil model větve (zbraň na blízko). Nejprve jsem jako potomka hlavní hráčovy kamery přidal objekt, který jsem nazval EquipmentPoint. Tento objekt představuje místo, kde má hráč zbraň držet. Jelikož je tento objekt potomek hlavní kamery, bude vždy umístěn relativně podle toho, kam se hráč zrovna dívá. Tomuto objektu jsem následně přidal další tři potomky (jednoho pro každý slot na zbraň). Jelikož jdou zbraně přepínat, bylo lepší zvolit tento postup, aby hra nemusela při každé změně zbraně vytvářet novou instanci 3D modelu. Místo toho je vždy aktivní pouze jeden slot se zbraní a ostatní sice stále existují, ale nejsou vidět.

4.5.8 Tvorba ukazatelů životních funkcí

Mnou již vytvořená logika konzumace předmětů zatím pouze vypisovala zprávu do konzole. Cílem tohoto úkolu však bylo, aby po konzumaci předmětu byly ovlivněny i hlavní hráčovy životní funkce. Tyto funkce jsem měl vyřešit celkem 3: žízeň, hlad a celkový život. Každý konzumovatelný Item měl nastaveny parametry pro každou z těchto životních funkcí (např. láhev s vodou sníží žízeň, maso sníží hlad, lékárnička doplní život).

Hodnoty těchto životních funkcí měly být graficky znázorněny v pravém horním rohu obrazovky. Do tohoto rohu jsem přidal základní obdélníkový obrázek, který jsem pouze vyplnil červenou barvou. Tento obdélník jsem pak přidal jako potomka objektu HealthBar. Následně jsem na tento objekt připnul komponent Slider, kterému jsem jako ovládaný objekt přiřadil vytvořený obrázek. Díky toho může Slider ovládat výplň obrázku jak ručně v editoru, tak přímo v kódu.

Z celého objektu HealthBar jsem vytvořil Prefab a jeho další 2 instance vložil do scény. Každé instanci jsem pak pouze změnil, jak často se má v čase měnit výplň. Jak je vidět na obrázku 4.2, abych se alespoň trochu přiblížil požadovanému vzhledu, předělal jsem nakonec základní obdélníkový obrázek na kruh

Obrázek 4.2: Ukazatele životních funkcí v rohu obrazovky

4.6 Hlavní herní menu

Smyslem této části bylo vytvořit hlavní herní menu. Jedná se o menu, které se hráči objeví hned po samotném zapnutí hry. Zde jsem měl vytvořit možnost hru zapnout nebo vypnout.

4.6.1 Využití řešení TextMeshPro

TextMeshPro představuje skvělé textové řešení pro Unity. Je to perfektní nástupce starého UI Textu a komponentu Text Mesh. TextMeshPro je velmi silný a jednoduchý k použití. Nabízí pokročilé textové vykreslovací techniky spolu s velkou řadou vlastních shaderů. Dává tak uživatelům obrovskou flexibilitu při stylování textu a jeho texturování. [3]

4.6.2 Scéna herního menu

Herní menu je vedeno ve 3D prostředí. Hráč tak na pozadí vidí scénu se stejnými modely, jaké jsou použity ve hře samotné. Přes tuto scénu se pak vykreslí uživatelské rozhraní se dvěma tlačítky, které slouží k zapnutí nebo vypnutí hry. K vytvoření scény jsem pouze přesunul kameru do správné polohy a před ní naskládal poskytnuté 3D modely. Vše jsem pak pouze upravoval tak, aby celý prostor působil přirozeně a nebyly vidět žádná rušivá prázdná místa. Zároveň jsem tento problém musel řešit tak, aby bylo použito co nejméně modelů. To bylo potřeba hlavně proto, aby se herní menu načítalo co možná nejkratší dobu a pohyb v něm nebyl příliš náročný na výkon počítače.

4.6.3 Tlačítka v herním menu

Do scény jsem na nový Canvas přidal pouze dvě tlačítka. Unity na svých UI tlačítkách využívá pouze obyčejný text, který se vykresluje jen ve velmi základním režimu a některé fonty zde nepůsobí hezky. Nahradil jsem proto tento text vylepšeným řešením pro text TextMeshPro. Tuto záměnu jsem si mohl dovolit, protože ve výsledku se s oběma druhy textu pracuje úplně stejně, akorát později jmenovaný má rozsáhlejší možnosti grafických úprav. Samotnou základní grafiku tlačítka jsem odstranil, protože nevypadá hezky. Pouze jsem přenastavil zbarvení pozadí tlačítka při najetí myší. Tlačítko je tak ze základů průhledné a jeho pozadí se zobrazí až ve chvíli, kdy je na něj najeto myší.

4.6.4 Přepínání scény v herním menu

Pro skutečné zapnutí hry bylo potřeba po stisknutí tlačítka "Play"přepnout na scénu prvního levelu. Na to jsem si vytvořil vlastní skript MainMenu, který využívám pouze ve scéně s herním menu. Do toho skriptu jsem nejprve připsal funkci, kde se přepne na zvolenou scénu podle jejího indexu. Tyto indexy jdou jednoduše zjistit přímo v enginu. Při otevření možnosti "Build menu"se zobrazí i seznam scén, které výsledný build obsahuje. Pořadí těchto scén pak zároveň značí i jejich index.

Podobně pak lze měnic scény i podle jejich názvu. Po stisknutí tlačítka "Quit"se hra ukončí. To zajistí funkce Quit přímo od Unity.

4.7 Systém audia v pozadí

Zvuky jsou ve hře velmi důležitou součástí a rozhodl jsem se tedy vytvořit management audia ještě před animacemi a dalšími úkoly. Zaměřil jsem se především na tvorbu zvuků v okolí a pozadí, protože jejich zdrojové soubory by se měly měnit dynamicky podle toho, kde a v jaké situaci se hráč zrovna nachází.

4.7.1 Komponent Audio Source

Komponent Audio Source přehraje zvukový soubor ve scéně. Audio Source může přehrát jakýkoli typ zvuku a může být nakonfigurován tak, aby tyto zvuky přehrával 2D, 3D nebo jejich kombinací. [3] Slyšitelná vzdálenost může být také ovládána pomocí křivek. Na každý Audio Source lze aplikovat zvukové efekty.

4.7.2 Vlastní skript AudioManager

V hierarchii jsem si vytvořil prázdný objekt s názvem AudioManager a přidal na něj skript stejného jména. Abych měl větší kontrolu nad jednotlivými zvuky, rozhodl jsem se pro vytvoření vlastní třídy Sound. V této třídě jsem vytvořil několik proměnných, které jsou pro kontrolu jednotlivých zvuků důležité.

Mým momentálním úkolem bylo promyslet systém tak, aby se mohlo přehrávat více zvuků najednou a bylo možné je jednotlivě pomalu utišit, aby jejich vypnutí působilo přirozeněji. Jelikož se jedná o zvuky v pozadí, které nemají žádný jasný zdroj, došel jsem k tomu, že bude potřeba využít několik současně aktivních komponent Audio Source na mém objektu AudioManager. Nejprve mě napadlo pro každý hrající zvuk vytvořit nový Audio Source komponent. Od tohoto postupu jsem byl však nakonec odrazen, protože je neefektivní a mohl by působit problémy za běhu hry. Rovněž mě jeden z programátorů upozornil, že více komponentů Audio Source na jednom objektu není moc dobrý nápad, ale v této konkrétní situaci se tomu bohužel nevyhnu. Ideální situací bylo tedy použít co nejméně těchto komponent. Nakonec jsem zvolil tři, protože více zvuků najednou v pozadí nikdy hrát nebude.

Ve skriptu AudioManager jsem si vytvořil pole typu Sound, kde se drží všechny často používané zvuky. Myšlenka byla taková, že by tyto zvuky mohly být spuštěny odkudkoli jen pomocí názvu. Do tohoto pole jsem přímo v editoru přidal dvě položky – zvuk lesa a zvuk u vody. Ve skriptu jsem si vytvořil pomocnou funkci, která vrátila vždy první volný Audio Source ze seznamu. Tato funkce se poté volala vždy, když měl být přehrán nějaký zvuk. Samotný kód pro přehrání zvuku zachycuje výpis 4.4. Zvuku byl pomocí této funkce přiřazen Audio Source a ten byl přidán do pole aktivních

zdrojů. Dále jsem zařídil, že zvuky jdou pomocí svého jména i vypnout. V tomto případě funkce pro vypnutí vyhledá v poli aktivních zdrojů konkrétní Audio Source, vypne jeho přehrávání a z pole aktivních zdrojů jej odebere. Takový Audio Source je pak opět brán jako neaktivní a může být znovu použit.

Požadovanou funkci FadeOut (zmizení zvuku do pozadí) jsem vyřešil tak, že jsem na daném komponentu Audio Source pouze snižoval hlasitost postupně do nuly. Na totožném principu jsem vytvořil i funkci FadeIn (opak funkce FadeOut), kde se audio postupně zvyšovalo.

```
public void Play(string name)
{
    Sound s = Array.Find(sounds, sound => sound.name == name);
    if (s != null)
    {
        s.source = GetEmptySource();
        if(s.source != null)
        {
            s.source.clip = s.clip;
            s.source.volume = s.volume;
            s.source.loop = s.loop;
            activeSources.Add(name, s.source); //add source to active sources
            s.source.Play();
        }
    }
    else { return; }
}
```

Listing 4.4: Ukázka kódu funkce pro přehrání zvuku

4.7.3 Změna zvuků při průchodu lokacemi

Ke konkrétním oblastem na herní mapě se měly vázat i konkrétní zvuky. Musel jsem zajistit, aby se v pozadí začal přehrávat určitý zvuk při vstupu do každé této oblasti a aby se přestal přehrávat, pokud hráč odešel. Pro přirozený začátek a konec zvuku jsem již měl naprogramovány funkce FadeIn a FadeOut, ale neměl jsem žádný způsob, jak změnu oblasti detekovat.

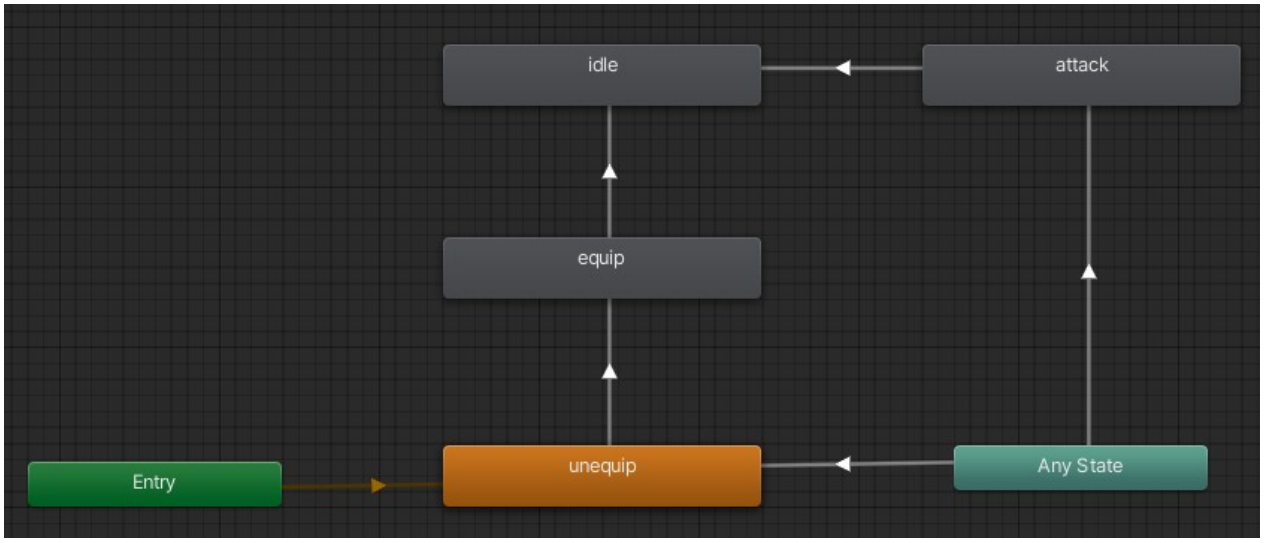
Pro tento úkol stačilo vytvořit prázdný objekt s jedním komponentem Sphere Collider, který uměl detekovat, jestli do něj hráč vstoupil. Pokud ano, začal přirozeně přehrávat daný zvuk. Při odchodu daný zvuk zase vypnul.

4.8 Tvorba zbraní a jejich animace

V této části bylo mým úkolem vyzkoušet si naprogramovat střelbu z revolveru. Na doporučení vedoucího jsem si pak měl vyzkoušet udělat i jednoduché animace držené zbraně.

Pro vykonání střelby jsem se rozhodl využít Raycast. Tento systém funguje tak, že z libovolného místa vystřelí neviditelný paprsek do libovolného směru. Díky tomuto paprsku je pak možné určit, do kterého objektu po cestě narazil. Pokud tedy Raycast vyslaný ze zbraně narazí do nepřítele, měl by být nepřítel zraněn. Pro test jsem ve scéně vytvořil dva kvádry, které měly představovat nepřítele. Ve výsledné hře bude tento skript připnut na divoká zvířata. Na testovací objekty jsem připnul svůj vytvořený skript Enemy, kde jsem přidal proměnnou znázorňující život nepřítele. Pokud tento život klesne na (nebo pod) hodnotu 0, nepřítel zemře.

Pro vytvoření animací jsem využil komponent *Animator*. Potřeboval jsem vytvořit animaci pro vytažení zbraně, útok a nečinnost. Nemusel jsem dělat nic složitého, stačilo pouze hýbat se zbraní. Po vytvoření všech animací jsem je všechny vložit do komponentu Animator. Jak je vidět na obrázku 4.3, z těchto stavů jsem pak vytvořil automat i s přechody. V Animatoru jsem pak vytvořil tři spouštěče, na základě kterých se rozhodovalo, který přechod se má provést.



Obrázek 4.3: Stavový automat v komponentu Animator

Celý tento postup platil pouze pro animování jedné konkrétní zbraně. Potřeboval jsem ale vytvořit ještě jednu zbraň na krátkou vzdálenost. Po chvilce čtení dokumentace jsem zjistil, že existuje komponent *Animator Override Controller*, díky kterého mohu využít již vytvořený stavový automat a pouze přepsat konkrétní animace. To se mi velice hodilo, protože útok zbraní na blízko má logiku animací úplně stejnou. Stačilo mi tak vytvořit pouze odlišné animace

Dalším požadavkem na střelnou zbraň byla její hlasitost. Měl jsem k dispozici zvuk výstřelu i s ozvěnou. Jelikož měl být původem zvuku právě držený revolver, jednalo se o lokální 3D zvuk a

využití mnou již vytvořeného skriptu AudioManager tak nedávalo moc smysl. Místo toho jsem pouze připnul na objekt revolveru komponent Audio Source a jako zvukový soubor zvolil zvuk výstřelu. Hlasitost přehrávání jsem pak zvýšil na úplně maximální úroveň.

4.9 Skript na střídání dne a noci

Nejdůležitějším faktorem při tvorbě denního cyklu byl pohyb slunce a celkové nasvětlení scény. Vytvořil jsem si proto skript LightingController, kde mělo být možné tyto dvě vlastnosti hry ovládat. Jako slunce slouží hlavní zdroj světla ve scéně. Stačilo tedy s tímto světlem pomalu otáčet a tím zajistit efekt západu a východu slunce. Ve skriptu jsem si vytvořil proměnnou, která šla nastavit v rozmezí 0 - 24. Na základě hodnoty této proměnné se pak vypočítala konečná poloha světla.

Na ovládání celkového zbarvení scény jsem opět využil Scriptable Object, abych mohl využívat více předvoleb světla (dále LightingPreset). Do této předvolby jsem dal tři barevné přechody – ambient color, fog color, directional color. Každá z těchto tří proměnných reprezentuje jednu část celkového nasvětlení scény. Pro určení, jaké barvy se mají zrovna aplikovat, jsem využil již vytvořenou proměnnou na otáčení slunce.

4.10 Tvorba změn počasí

Jelikož se mělo jednat o prototyp hry se survival prvky, muselo být vytvořeno i střídání počasí. Dostal jsem tedy za úkol vytvořit systém, který by náhodně střídal různé druhy počasí. Nejdůležitější složkou změny počasí je Skybox, což je vlastně objekt, který je vykreslen okolo celé scény a dává tak pocit rozsáhlého světa i za horizontem. [3] Jeho nejčastější využití je pro práci s oblohou.

Pro slunečné počasí nebylo potřeba skoro nic tvořit, protože scéna už je od základu postavena tak, že je slunečno. Pouze jsem si malinko upravil Skybox, aby více odpovídal celkové grafické představě. Tento Skybox jsem si uložil jako slunečný Skybox.

Tvorba deště už však byla trochu náročnější. Nestačilo jen upravit světelné podmínky, ale také přidat samotné kapky deště. Vytvořil jsem si tedy Particle System, který z určité plochy začal generovat protáhlé bílé částice. Jejich počet a rychlost jsem postupně upravil, ať déšť vypadá přirozeně. Z důvodu lepšího výkonu navíc bylo žádoucí, aby se částice negenerovaly nad celou mapou, ale jen v okolí hráče. Abych mohl jednoduše počasí měnit, využil jsem znovu Scriptable Object, abych si vytvořil předvolbu pro počasí. V této předvolbě jsem vytvořil proměnné pro odpovídající LightingPreset, Particle System (pokud byl nějaký potřeba, u deštně např. již zmíněné kapky deště), zvuk v pozadí a Skybox.

4.10.1 Skript na změnu počasí

Aby vše zapadlo do sebe, musel jsem vytvořit skript WeatherController, kde by se počasí měnilo. Jako první byla do skriptu přidána kolekce na jednotlivé předvolby počasí a k ní i funkce, co vždy

vybere jednu z nich. Pokud se bude lišit od aktuálního počasí, předvolba se aplikuje. Tato aplikace využila můj LightingController ze sekce 4.9 na změnu světla a případě do scény vložila požadovaný Particle System.

4.10.2 Plynulý přechod barev při změně počasí

Problém s mým vytvořeným systémem byl v tom, že když se změnilo počasí, změnily se okamžitě světelné podmínky i Skybox, což působilo nepřirozeně. Místo toho mi vedoucí navrhnul, abych využíval pouze jeden Skybox, ale postupně vždy měnil jeho vlastnosti. Skript LightingPreset jsem přepsal tak, aby neměnil barvy okamžitě, ale místo toho použil funkci Color.Lepr a pro číselné hodnoty Math.Lepr. Tyto dvě funkce zajišťují plynulý přechod mezi dvěma hodnotami ve zvoleném časovém intervalu. Jejich využití v mém kódu lze vidět ve výpisu 4.5.

```
private void UpdateLighting(float currentTime)
{
    RenderSettings.ambientLight = Color.Lerp(RenderSettings.ambientLight, Preset.
        AmbientColor.Evaluate(currentTime), 0.007f);
    RenderSettings.fogColor = Preset.FogColor.Evaluate(currentTime);

    if(DirectionalLight != null)
    {
        DirectionalLight.color = Color.Lerp(DirectionalLight.color, Preset.
            DirectionalColor.Evaluate(currentTime), 0.007f);
        DirectionalLight.transform.localRotation = Quaternion.Euler(new Vector3((
            currentTime * 360f) - 90f - 170f, 0));
    }
}
```

Listing 4.5: Ukázka kódu funkce pro plynulou změnu počasí

4.11 Generování terénu

Pro účely práce s terénem jsem si z oficiálních pluginů doinstaloval nástroje pro tvorbu terénu. Pomocí těchto nástrojů je možné terén podle libosti tvarovat, přidávat detaily nebo jej obarvit texturami.

4.11.1 Tvorba terénu pomocí výškové mapy

Pro generování terénu je možné využít i výškovou mapu. Jedná se o obrázek s různými odstíny šedé barvy. Podle světlosti barvy pak nástroje pro terén vytvoří požadovaný tvar (čím světlejší, tím vyšší

terén). Pomocí těchto map je velmi jednoduché vytvořit i složitý terén velmi rychle. Jelikož jsem takovou mapu měl k dispozici, nemusel jsem terén tvarovat ručně. Stačilo mi pouze importovat tuto mapu do Unity a použít ji jako výškovou mapu v nástrojích pro tvorbu terénu.

4.11.2 Pokládání stromů na vytvořený terén

K dispozici jsem měl několik modelů stromů, které jsem měl na již vytvořený terén rozmístit. Žádný z těchto modelů ještě neměl kolize. Musel jsem všechny modely upravit a kolize jim přidat, jinak by jimi hráč mohl procházet.

Nejprve mě napadlo rozmístit stromy ručně, ale to by bylo velmi zdlouhavé a neefektivní. Nástroje pro tvorbu terénu naštěstí nabízí nástroj pro masivní rozmístění stromů. Jako první bylo potřeba umístit poskytnuté modely do seznamu stromů. Po vložení modelů do seznamu stromů je dále nutné určit, kolik stromů se má na terén umístit, v jaké vzdálenosti mají být a do jaké výšky je možné je umístit. Hromadné položení stromů se pak provede stisknutím tlačítka v editoru.

Díky nástrojům pro tvorbu terénu je navíc možné jednotlivé stromy upravovat pomocí štětců. Po zvolení této možnosti v editoru lze kreslením na terén stromy dodatečně přidávat nebo naopak mazat.

4.12 Vytvoření zvířete pomocí navigačního systému v enginu Unity

Navigační systém umožňuje vytvořit postavy, které se dokážou volně pohybovat v herním světě. Tento systém dává postavám např. schopnost pochopit, že pro přístup na druhé patro potřebují vyjít po schodech nebo třeba přeskočit díru. Celá funkce tohoto systému se skládá z několika částí [3]:

- **Navigation Mesh** – jedná se o datovou strukturu, která popisuje dosažitelné povrchy herního světa a umožňuje najít cestu z jedné lokace do druhé. Tato struktura se dá poměrně jednoduše vytvořit na již hotové scéně.
- **Navigation Mesh Agent** – komponent, který dokáže posouvat postavy z jednoho místa na druhé tak, aby do sebe nenarážely. K určení směru pohybu využívá právě Navigation Mesh.
- **Off-Mesh Link** – tento komponent dokáže ve scéně vytvořit zkratky, které běžně nedokáže popsat Navigation Mesh. Jedná se tedy např. o místa, kde je nutné něco přeskočit.
- **Navigation Mesh Obstacle** – jde o komponent, díky němož lze určit, kterým objektům se má Agent vyhnout. Tento komponent se používá hlavně pro pohyblivé objekty

4.12.1 Navigation Mesh v již přichystaném terénu

Abych mohl vůbec nějaké zvíře do hry přidat, musel jsem nejprve vytvořit Navigation Mesh na již vygenerovaný terén. Vrátil jsem se tedy na mou testovací scénu, která je výrazně menší než finální

terén. Generování navigační struktury zde trvalo vždy jen velmi krátkou dobu, a proto jsem mohl nastavení měnit tak dlouho, dokud se struktura nezdála dostačující. Unity vždy vygenerovanou navigační plochu označí modrou barvou a díky toho jsem výsledek mé konfigurace mohl vidět vždy okamžitě po generování.

4.12.2 Pohyb zvířete pomocí komponentu Navigation Mesh Agent

Po vygenerování navigační struktury jsem mohl přistoupit k samotnému pohybu zvířete. K tomuto úkolu jsem měl k dispozici model zvířete i s animacemi. Jeho chování však bylo nutné doprogramovat. Objekt zvířete jsem tedy přidal do scény a připnul mu komponent *Navigation Mesh Agent* (dále jen Agent), aby uměl spolupracovat s vytvořenou navigační strukturou.

Abych zvíře opravdu rozpohyboval pomocí navigačního komponentu, vytvořil jsem si vlastní skript. Začal jsem tím, že tento skript vždy našel novou pozici, kam by se měl objekt posunout. Jak je vidět ve výpisu 4.6, tato funkce v okolí zvířete najde bod a zkontroluje, jestli se nachází na herní ploše. Tento bod je vždy nejprve nalezen nad samotným terénem, odkud je pak vygenerován Raycast směrem dolů. V bodě dopadu je nalezen nový bod a ten je předán Agentovi jako nová destinace.

```
private void FindNextWalkPoint()
{
    float z = Random.Range(-walkPointRange, walkPointRange);
    float x = Random.Range(-walkPointRange, walkPointRange);

    walkPoint = new Vector3(transform.position.x + x, transform.position.y,
        transform.position.z + z);

    if(Physics.Raycast(walkPoint, -transform.up, 2f, groundLayer))
    {
        nextWalkPointSet = true;
    }
}
```

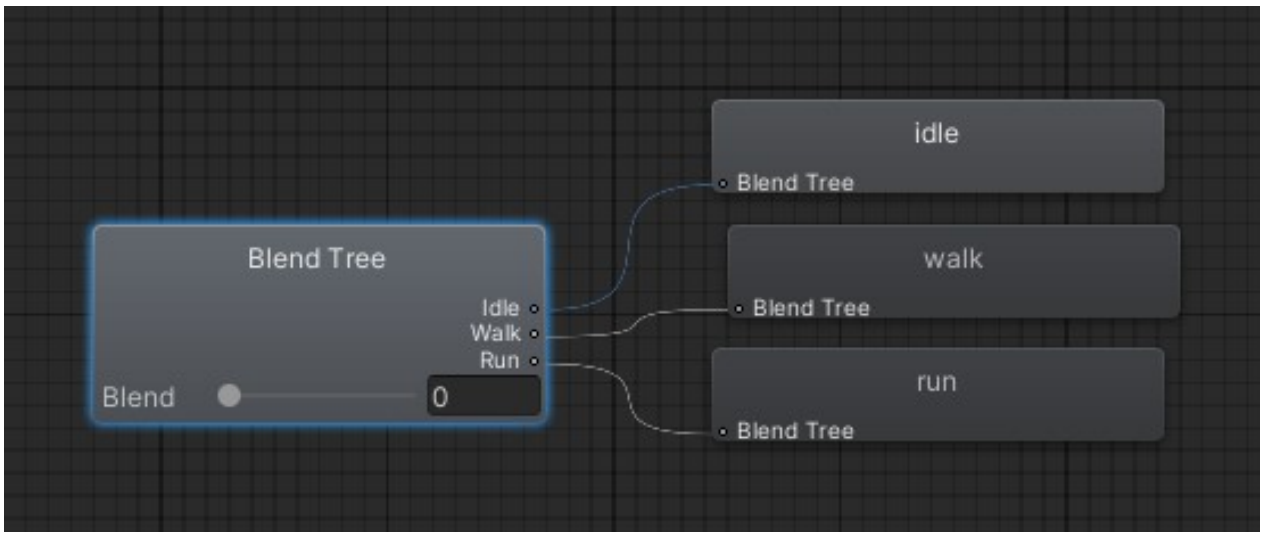
Listing 4.6: Ukázka kódu funkce pro nalezení další destinace Agenta

4.12.3 Animace pohybu zvířete

K logickému navázání několika pohybových animací je klasické vytváření stavového automatu nevhodné, protože by následný kód pro ovládání tohoto automatu nebyl přehledný. Místo toho může být využit *Blend Tree*, což je struktura, která dokáže plynule přecházet mezi animacemi jen na základě hodnoty jedné proměnné.

Vytvořil jsem si tedy prázdný Blend Tree a vložil do něj tři animace – chůze, běh a nečinnost. Pro každou z animací jsem připsal její maximální hodnotu řídící proměnné. Pokud byla tato hodnota překročena, začala se plynule přehrávat animace rychlejšího pohybu. Dokázal jsem tedy plynule přejít z pomalé chodící animace na rychlejší animaci při pronásledování hráče. Na tento Blend Tree jsem nakonec napojil animaci pro útok. Ta už se spouštěla přechodem ze samotného Blend Tree pomocí spouštěče. Podoba mnou vytvořeného Blend Tree přímo v editoru je vidět na obrázku 4.4.

Následně bylo zvířeti potřeba doprogramovat chování a mezi animacemi přepínat přímo v kódu. Přidal jsem dvě proměnné, které značily vzdálenosti. První vzdálenost říká, jak daleko zvíře vidí. Pokud se hráč dostane do této vzdálenosti, začne jej zvíře pronásledovat. To zároveň spustí animaci běhu a dalším bodem Agenta se stane právě objekt hráče. Tak to bude až do té doby, dokud se hráč nedostane z dosahu zvířete. Druhá vzdálenost je velmi malá a značí, jak daleko hráč musí být, aby na něj zvíře zaútočilo. Při útoku se spustila animace útoku a navíc udělilo zvíře hráčovi poškození.



Obrázek 4.4: Blend Tree se třemi stavy

Kapitola 5

Závěr

5.1 Dosažené výsledky v průběhu praxe

Od začátku svého působení ve firmě TastyAir s.r.o. jsem vytvořil hratelný prototyp Survival hry tak, jak se píše v design dokumentu. Hra momentálně obsahuje všechny důležité pohybové a další gameplay mechaniky. Zajímavý je především systém inventáře, nad kterým bylo uděláno opravdu hodně práce a výsledkem je, že má hráč velkou volnost v práci se svým vybavením. Do svého inventáře může nasbírat velké množství různých předmětů. Z některých lze poskládat jiné předměty, jiné je možné vzít do ruky a další lze zase konzumovat. Hráč si musí hlídat své životní funkce, což jej nutí k dalšímu prozkoumávání herního světa.

Aby hra působila živěji, je vytvořen cyklus dne a noci a také systém počasí. Tento systém je vytvořen tak, aby bylo možné i v budoucnu přidat nové druhy počasí. Každé počasí má své nasvětlení podle momentálního herního času. Počasí a celkové světelné podmínky se navíc mění plynule a hráče tak tyto změny nepříjemně nevyrušují.

Dále jsem vytvořil plně funkční zvíře, které se libovolně prochází po herním světě a může na hráče zaútočit.

5.2 Možná vylepšení

Na projektu je stále potřeba pracovat zejména po grafické stránce. Zatím chybí většina vizuálních efektů, na kterých by se však mělo pracovat spíše v jedné z posledních fází vývoje. Dále je potřeba implementovat systém dialogů a úkolů, díky kterým bude možné vyprávět herní příběh. Mnou vytvořená mapa představuje pouze část herního světa a v budoucnu bude tak nutné vytvořit další levely a scény.

5.3 Získané zkušenosti a dovednosti

Tato praxe mi pomohla seznámit se hlavně s tím, jak skutečně probíhá vývoj her a získal jsem pojem i o jeho časové náročnosti. Při práci na budoucích projektech tak budu schopen lépe určit potřebný čas na jejich dokončení. Rovněž jsem získal spoustu zkušeností s herním enginem Unity, se kterým jsem se za dobu svého studia nesetkal vůbec. V průběhu praxe jsem se také naučil používat systém Git pro správu verzí.

Při své práci jsem využil rozsáhlých znalostí jazyka C# a navíc jsem se naučil jeho zajímavá rozšíření, která engine Unity nabízí. Dále jsem využil znalosti z předmětů, které se zabývaly objektově orientovaným programováním. Rovněž jsem zužitkoval znalosti z předmětu Softwarové Inženýrství hlavně při plánování postupu své práce.

Praxe mi dále pomohla si uvědomit, jak je důležité nad kódem přemýšlet a všechny fáze vývoje nejdříve dobře naplánovat.

Literatura

[1] Game engine - Wikipedia. [online]. [cit. 15.12.2020]. Dostupné z: https://en.wikipedia.org/wiki/Game_engine

[2] Best Game Engines of 2021. GameDev Academy – Tutorials on Game Development, Unity [online]. Copyright © 2021 [cit. 17.12.2020]. Dostupné z: https://gamedevacademy.org/best-game-engines/

[3] Unity - Manual: Unity User Manual 2020.3 (LTS). [online]. Copyright © 2020 Unity Technologies. [cit. 27.01.2021]. Dostupné z: https://docs.unity3d.com/Manual/

[4] Unity - Scripting API: . [online]. Copyright © 2020 Unity Technologies. [cit. 09.01.2021]. Dostupné z: https://docs.unity3d.com/ScriptReference/

[5] Tasty Air - Virtuální a Rozšířená realita pro podniky a firmy [online]. Copyright © 2019 [cit. 02.04.2021]. Dostupné z: https://www.tastyair.cz/

[6] HOLAN, Tomáš. Unity: první seznámení s tvorbou počítačových her. Praha: CZ.NIC, z.s.p.o., 2020. CZ.NIC. ISBN 978-80-88168-57-7.